2-Month Roadmap (8 Weeks) — Advanced Beginner → Junior Malware Analyst

Daily Commitment: ~3–4 hours focused practice

Tools / Environment:

- VM setup: VirtualBox or VMware Workstation
- Debugging: WinDbg, x64dbg
- Reverse engineering: Ghidra, IDA Pro
- Analysis: PE Studio, Process Monitor

Week 1 — Advanced PE & Windows Internals

Goal: Master PE structure and Windows process internals.

Tasks:

1. Study PE headers in depth: DOS header, NT headers, sections, imports/exports.
2. Practice mapping addresses manually to find entry points.
3. Inspect loaded modules and memory layout of a process in WinDbg.
4. Examine PEB and TEB structures of a running process.

Exercise: Take a small `.exe` (like Notepad), map all imports manually, and locate the entry point in memory.

Week 2 — Manual API Resolution & Process Injection Basics

Goal: Start manipulating processes safely.

Tasks:

1. Launch a process and inject shellcode manually using VirtualAlloc/WriteProcessMemory/CreateRemoteThread.
2. Practice API hashing and manual resolving techniques.
3. Understand process rights and handle tables.

Exercise: Inject a small payload into `notepad.exe` to launch `calc.exe` dynamically.

Week 3 — Debugging & Anti-Debug Awareness

Goal: Level up WinDbg skills and understand anti-analysis techniques.

Tasks:

1. Set conditional breakpoints, watchpoints, and memory breakpoints.
2. Explore anti-debug techniques: IsDebuggerPresent, CheckRemoteDebuggerPresent, timing checks.
3. Practice bypassing simple anti-debug checks in controlled samples.

Exercise: Modify a simple sample to bypass an anti-debug check and step through it in WinDbg.

Week 4 — Shellcode & Exploitation Fundamentals

Goal: Write and execute simple shellcode in a controlled lab.

Tasks:

1. Write position-independent shellcode in assembly for Windows (launch calc.exe, reverse shell).
2. Execute shellcode in memory via manual mapping.
3. Practice building a basic ROP chain in a small C program.

Exercise: Inject your shellcode into a running process from Week 2 without touching disk.

Week 5 — Static Malware Analysis

Goal: Analyze binaries without running them.

Tasks:

1. Use PE Studio and Ghidra/IDA Pro to identify suspicious sections, imports, and strings.
2. Reverse simple malware samples from safe sources (crackmes, test malware).
3. Document indicators of compromise (IOCs): hashes, API calls, suspicious sections.

Exercise: Pick 2–3 small malware samples, create a written analysis report.

Week 6 — Dynamic Malware Analysis

Goal: Analyze malware behavior safely in a VM.

Tasks:

1. Run samples in an isolated VM and monitor using:
    - Process Monitor
    - Process Explorer
    - Wireshark
2. Record process creation, registry changes, network activity.
3. Compare static vs dynamic observations.

Exercise: Document behavior of a single sample (create a mini report).

Week 7 — Advanced Techniques & Malware Tricks

Goal: Understand common malware evasion and injection techniques.

Topics to explore:

1. Process hollowing
2. Reflective DLL injection
3. APC injection / thread hijacking
4. Anti-VM & anti-sandbox checks

Exercise: Try to identify these techniques in small proof-of-concept samples.

Week 8 — Portfolio & Real-World Preparation

Goal: Build a visible portfolio for junior roles.

Tasks:

1. Compile analysis reports from Weeks 5–7.
2. Publish sanitized reports on GitHub or personal blog.
3. Complete a few CTF-style reverse engineering challenges (Crackmes.one, Exploit Education).
4. Practice writing clear technical reports like a junior malware analyst would.

Exercise: Create a “Junior Malware Analyst Portfolio” with 3–5 analysis reports and lab exercises.

✅ Expected Outcome After 2 Months

By the end of this roadmap, you should be able to:

- Reverse-engineer small Windows binaries
- Analyze malware statically and dynamically
- Write and inject simple shellcode
- Detect anti-debug and basic anti-analysis techniques
- Build a **portfolio demonstrating real skills**
- Be **job-ready for remote junior malware analyst / reverse engineer roles**

Perfect! Since you want a **complete, step-by-step 60-day roadmap “without missing anything”** for someone who already knows **C, assembly, WinDbg, manual Windows API resolving, and basic shellcode (launching calc.exe)**, I’ve put together a **day-by-day plan** that covers **every critical skill you need for junior malware analyst / reverse engineer roles**. This roadmap ensures **nothing essential is skipped**.

2-Month (60-Day) Complete Roadmap

Goal: Job-ready for junior malware analyst / reverse engineer roles (remote or WFH).

Daily Commitment: ~3–4 hours focused practice

Tools / Environment:

- VM lab: VirtualBox or VMware Workstation
- Debugging: WinDbg, x64dbg
- Reverse Engineering: Ghidra, IDA Pro
- Analysis: PE Studio, Process Monitor, Process Explorer, Wireshark

---

Week 1 — Deep Windows Internals & PE Structure

Day 1–2:

- Review Windows process & thread structures
- Explore memory layout (stack, heap, code, data segments) in WinDbg

Day 3–4:

- Analyze PEB, TEB, and VAD in memory
- Map Notepad.exe and view loaded DLLs

Day 5–6:

- Study PE headers in depth (DOS Header, NT Headers, Section Table)
- Identify Import/Export tables

Day 7:

- Practice mapping functions manually in a small EXE

Week 2 — Manual API Resolution & Injection Prep

Day 8–9:

- Recreate API resolution in memory manually (LoadLibrary + GetProcAddress)

Day 10–11:

- Allocate memory in remote process with VirtualAlloc
- WriteProcessMemory & CreateRemoteThread

Day 12–13:

- Implement simple shellcode injection (launch calc.exe)
- Explore process rights and handle tables

Day 14:

- Document injection process with diagrams & lab notes

Week 3 — Advanced Debugging & Anti-Debug Techniques

Day 15–16:

- Breakpoints: conditional, memory, hardware breakpoints in WinDbg/x64dbg

Day 17–18:

- Explore anti-debugging checks: IsDebuggerPresent, CheckRemoteDebuggerPresent

Day 19–20:

- Step through small samples with anti-debug tricks
- Bypass timing-based checks

Day 21:

- Lab: Inject shellcode into a process with anti-debug check bypassed

---

Week 4 — Shellcode & Exploitation Basics

Day 22–23:

- Write position-independent shellcode (calc.exe, messagebox, reverse shell)

Day 24–25:

- Execute shellcode in-memory via manual mapping

Day 26–27:

- Learn basic stack/heap exploitation techniques (overflow demo in C)

Day 28:

- Lab: Combine shellcode with small buffer overflow demo in a safe environment

---

Week 5 — Static Malware Analysis

Day 29–30:

- Analyze PE files using PE Studio
- Check imports, sections, suspicious headers

Day 31–32:

- Reverse engineer small malware samples (crackmes) in Ghidra
- Trace function flow and API calls

Day 33–34:

- Document static indicators of compromise (IOCs)

Day 35:

- Lab: Full static analysis report for 1–2 small samples

Week 6 — Dynamic Malware Analysis

Day 36–37:

- Setup isolated VM with snapshot
- Monitor process behavior with Process Monitor / Process Explorer

Day 38–39:

- Capture network activity with Wireshark

Day 40–41:

- Correlate static and dynamic analysis findings

Day 42:

- Lab: Create detailed behavior report (file changes, registry, network)

Week 7 — Advanced Malware Techniques

Day 43–44:

- Process hollowing & reflective DLL injection (study examples)

Day 45–46:

- APC injection & thread hijacking

Day 47–48:

- Anti-VM, anti-sandbox detection techniques

Day 49:

- Lab: Identify these techniques in controlled POCs

Week 8 — Portfolio & Job Prep

Day 50–52:

- Compile 3–5 analysis reports (static + dynamic)
- Create GitHub or personal blog to host reports

Day 53–55:

- Solve 3–5 crackmes / CTF reverse challenges
- Include annotated solutions in portfolio

Day 56–58:

- Practice writing professional malware analysis reports (clear, concise)

Day 59–60:

- Review all lab exercises, check tools mastery (WinDbg, x64dbg, Ghidra, IDA Pro)
- Finalize portfolio ready for job applications

✅ End-of-60-Day Skills Snapshot

By completion, you will be able to:

- Reverse-engineer small Windows binaries efficiently
- Analyze malware **statically and dynamically**
- Write and execute **manual shellcode**
- Detect anti-debug / anti-VM techniques
- Understand **process injection, hollowing, DLL reflective loading**
- Build a **portfolio demonstrating real-world skills** ready for junior malware analyst / reverse engineering roles

---

If you want, I can also **map this 60-day plan to a "job-ready skill checklist for CrowdStrike / Kaspersky / Unit 42"**, showing exactly which tasks align with **junior hiring requirements**, so you can **apply immediately after 2 months**.

Do you want me to do that?

A proper **malware analysis lab setup** is extremely important. It must be **isolated, safe, and controlled** so malware cannot escape to your real system or internet. Below is a **complete Windows-focused lab setup used by beginner and junior malware analysts**. 🧪💻

---

1 Host Machine (Your Real Computer)

Your main system should be clean and only run virtualization software.

Recommended specs:

```
CPU: 4–8 cores
RAM: 16 GB (minimum 8 GB)
Storage: 200 GB free
```

Install virtualization software like:

- VirtualBox
  or
- VMware Workstation

These allow you to create **isolated virtual machines (VMs)**.

---

2 Analysis Virtual Machine

Create a **Windows VM** for analysis.

Recommended OS:

- Microsoft Windows (Windows 10 or 11)

VM configuration example:

```
RAM: 4–8 GB
CPU: 2–4 cores
Disk: 60–100 GB
Network: Host-only or Internal network
```

Important:
❗**Disable shared folders and clipboard initially** to avoid malware escaping.

---

3 Essential Analysis Tools (Install Inside VM)

These tools help you analyze binaries and malware behavior.

Debugging Tools

- WinDbg
- x64dbg

Used to:

```
step through assembly
set breakpoints
inspect registers
trace execution
```

Reverse Engineering Tools

- Ghidra
- IDA Pro

Used to:

```
disassemble binaries
view functions
trace program logic
```

Static Analysis Tools

- PE Studio
- Detect It Easy

These show:

```
PE headers
imports
```

```
packers
suspicious sections
```

Dynamic Monitoring Tools

Install behavior monitoring tools:

- Process Monitor
- Process Explorer
- Wireshark

These monitor:

```
file activity
registry changes
network connections
running processes
```

---

4 Snapshot System (Critical)

Before running malware:

Create a **VM snapshot**.

Snapshots allow you to:

```
run malware
observe behavior
restore VM instantly
```

Both virtualization platforms support this:

- VirtualBox
- VMware Workstation

---

5 Network Isolation

You must control the network.

Safe options:

```
Host-only network
```

```
Internal network
No internet access
```

If malware needs internet behavior testing, simulate network traffic locally.

---

6 Optional Fake Internet Simulation

Some malware expects internet servers.

Tools sometimes used in labs:

```
Fake DNS
Local HTTP server
Network simulation
```

This allows malware to run without contacting real attackers.

---

7 Folder Structure in Lab

Organize files clearly:

```
/malware_samples
/tools
/reports
/shellcode_tests
/crackmes
```

Keep malware samples separate from tools.

---

8 Safe Malware Sources for Practice

Never download random malware. Use controlled samples from:

- Malware Traffic Analysis
- Crackmes.one

These are widely used by malware analysts for learning.

9️⃣ Lab Workflow

Typical workflow in a malware lab:

```
1. Take VM snapshot
2. Perform static analysis
3. Run malware in VM
4. Monitor behavior
5. Debug with WinDbg/x64dbg
6. Document findings
7. Restore snapshot
```

🔟 Example Beginner Malware Lab Architecture

```
Host Computer
│
├── Virtual Machine (Windows 10)
│
├── Tools Installed
│    ├── WinDbg
│    ├── x64dbg
│    ├── Ghidra
│    ├── IDA Pro
│    ├── PE Studio
│    ├── Process Monitor
│    └── Wireshark
│
└── Malware Samples
```

⚠ Safety Rules (Very Important)

Always follow these rules:

```
Never run malware on host OS
Disable shared folders
Use VM snapshots
Isolate network
Keep host antivirus active
```

These rules protect your real system.

🎯 Result of Proper Lab Setup

With this lab you can safely practice:

- reverse engineering
- debugging binaries
- malware analysis
- shellcode testing
- process injection

These are the **same skills used by analysts at companies like CrowdStrike and Kaspersky.**

✅ If you want, I can also show you **a "professional malware lab architecture" used by elite researchers** (much more advanced, with **3–4 VMs, fake internet, and monitoring systems**).

Absolutely — here's a **professional-grade malware lab architecture** that elite malware researchers and threat analysts use. This setup is designed for **safe, full-featured analysis**, including **dynamic, network, and multi-system interactions**, as well as **advanced monitoring**.

Professional Malware Lab Architecture (3–4 VMs + Fake Internet)

Goal: Safely analyze malware that uses network communication, multi-process injection, anti-VM checks, or sandbox evasion.

1️⃣ Host Machine (Your Base PC)

Requirements:

```
CPU: 8+ cores
RAM: 32 GB+ (16 GB minimum)
Storage: 1 TB+ SSD
OS: Windows 10/11 or Linux for orchestration
```

Software:

- VirtualBox or VMware Workstation

- Snapshot support enabled
- Optional: VMware ESXi / Proxmox for advanced lab orchestration

Purpose: Runs all VMs and manages network isolation.

---

2 VM Layout

Typically 3–4 VMs, each isolated:

VM	Purpose
Analysis VM	Windows 10/11, malware runs here. All monitoring and debugging tools installed.
Control VM	Simulates C2 servers (HTTP, HTTPS, TCP, DNS). Malware can "call home" safely.
Network Monitor VM	Linux or Windows, runs Wireshark, Zeek/Bro, Suricata for full traffic capture.
Optional Sandbox VM	Auto-analysis, scripts to observe malware in multiple scenarios or triggers.

---

3 Network Setup

Custom virtual network configuration:

```
- Internal network (VMs can talk only to each other)
- Host-only network for internet access control
- NAT disabled for malware VM
- Fake DNS and routing VM (simulate real internet endpoints)
```

Purpose: Malware thinks it has internet, but host is protected.

---

4 Tools Installed per VM

Analysis VM

- Debugging: WinDbg, x64dbg
- Reverse Engineering: Ghidra, IDA Pro
- Static Analysis: PE Studio, Detect It Easy

- Dynamic Analysis: Process Monitor, Process Explorer
- Optional: Sandbox automation tools (Cuckoo Sandbox)

Control VM (Fake Internet/C2)

- Web server (Apache/Nginx)
- FTP/TFTP server
- Fake DNS server
- Optional: Metasploit multi-handler (for reverse shells)

Network Monitor VM

- Wireshark (full packet capture)
- Zeek/Bro (network analysis)
- Suricata (IDS/alerting)

---

5 Malware Analysis Workflow

1. **Snapshot the Analysis VM** – always revert before new sample.
2. **Static Analysis** – PE headers, strings, suspicious APIs.
3. **Dynamic Analysis** – run malware in Analysis VM.
4. **Network Monitoring** – capture traffic in Network Monitor VM.
5. **C2 Simulation** – Control VM handles DNS, HTTP, TCP requests.
6. **Debugging** – step through malware in WinDbg/x64dbg.
7. **Logging & Reporting** – store all findings in a separate “Reports” VM or host folder.
8. **Reset VM** – revert to snapshot after each experiment.

---

6 Safety & Security Practices

- Isolate malware VM networks completely from host.
- Do not share folders or clipboard with host.
- Use snapshots before and after running malware.
- Optional: run host antivirus in passive mode for extra containment.

---

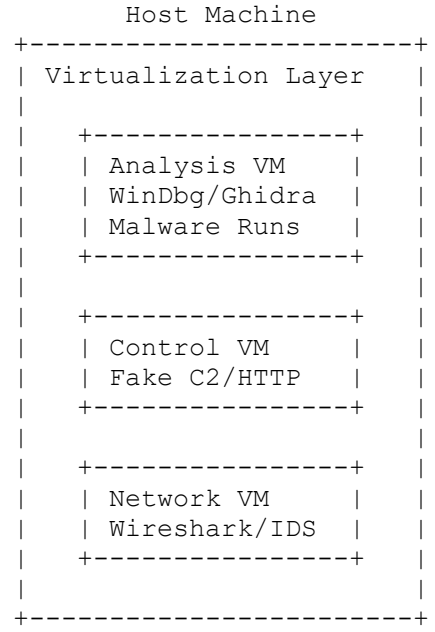
7 Optional Advanced Additions

- **Multiple Analysis VMs** – simulate enterprise endpoints

- **Automated scripts** – run malware, capture screenshots, monitor registry & filesystem changes
- **Full packet capture** – replay malware network behavior safely
- **Threat intelligence integration** – track malware families, IOCs, and hashes

---

Diagram Representation



This lab lets you safely:

- Run malware that communicates externally
- Analyze multi-process malware and injection techniques
- Capture network traffic and detect stealthy behavior
- Debug and reverse engineer real malware professionally

I can also **create a ready-to-use lab configuration file with VM specs, IP addresses, and networking layout**, so you can replicate this **exact professional lab** without guessing settings.

Do you want me to make that next?